# COLLABORATIVE WEB BASED DEVELOPMENT INTERFACE

## CROSS REFERENCE TO RELATED APPLICATION

[0001] The present invention is related to a U.S. Patent Application No. 09/852,959 entitled "System and Method for Improving The Performance of a Web Application by Building Only Requested Web Pages Only in A Requested Human Language" to Brewer et al., filed May 10, 2001, published November 14, 2002 as Published Application No. 2002/0169802 and assigned to the assignee of the present invention.

## BACKGROUND OF THE INVENTION

### *Field of the Invention*

[0002] The present invention is related to web-based management interfaces and more particularly to a collaborative design system for designing web-based management interfaces.

### *Background Description*

[0003] Increasingly, network based devices include a web-based management interface that provides users with information such as system status. Since the system status information may be changing, constantly, a typical such web interface application is designed to dynamically build and rebuild new web pages with every information change. The typical interface may provide web pages built from a collection of hypertext mark up language (HTML) files with placeholders in markup text for receiving and displaying dynamic input data. An executable code module can generate variable data, which another executable code module receives and combines with the HTML template

files to produce complete HTML documents. The HTML documents are transferred to web-based clients using standard hypertext transport protocol (http) and displayed as web pages.

[0004] Typically, such an interface is a collaboratively designed by a number of groups with design responsibility for the HTML template files and executable code modules assigned to each of the groups. The development groups must maintain good communications during design. The executable code modules must designed to provide whatever variable data is necessary and in an acceptable format and, then, combine the data with the HTML template files in the correct placeholder locations to provide useful pages. Further, the web pages may include repeated item structures that are stored in system memory which must be locatable by the executable code modules. Frequent calls for those repeated structures can increase the size of the executable code, inefficiently using system resources and restricting HTML template file design, thereby impairing and/or impeding the coding effort. As a result, these design constraints may limit what tabular data can be incorporated into the final web pages or how many pages are available.

[0005] Thus, there is a need for an efficient collaborative design system for designing web-based management interfaces and for collecting variable data and passing collected data for display in web pages provided by the particular web-based management interface.

## SUMMARY OF THE INVENTION

[0006] It is a purpose of the invention to control executable code module size growth;

[0007] It is another purpose of the invention to improve the tabular data handling efficiency of web-based management interfaces;

[0008] It is yet another purpose of the invention to maintain good communications between design groups designing parts of a web-based management interfaces;

[0009] It is yet another purpose of the invention to increase HTML template file and web page design flexibility when incorporating tabular data into the web pages;

[0010] It is yet another purpose of the invention to optimize storage of repeated item structures and facilitate the location of such structures by executable code modules to improve system resource efficiency and to allow a more dynamic coding effort.

[0011] The present invention relates to a data structure for repeated data items presented in tabular lists in a web based management interface, to a collaborative web based management interface design system and to a system and program product including the data structure. The data structure includes a page pointer table with links to repeatable data structures and corresponding page maps. The page maps have links to tabular lists in corresponding repeatable data structures. The page pointer table may be included in executable code modules generating variable data and generating web pages from the data. Adding web pages to the interface only increases the size of the executable code modules by the size of a repeatable data structure link, a map page link and a map page length indicator.

## BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

[0013] Figure 1 shows an example of a collaborative system for designing web-based management interfaces according to a preferred embodiment of the present invention;

[0014] Figure 2 shows an example of how variable data was organized in a prior data structure;

[0015] Figure 3 shows an example of data organized and maintained in accordance with the present invention.

## DESCRIPTION OF PREFERRED EMBODIMENTS

[0016] Figure 1 shows an example of a collaborative system 100 for designing web-based management interfaces with design functions distributed amongst a number of design groups 102A, 102B, 102C according to a preferred embodiment of the present invention. The collaborative system has application to designing a web-based management interface for displaying system information on web pages such as for example, for the IBM Corporation's TotalStorage™ Enterprise Tape Library 3494. The collaborative system 100 includes a collection of hypertext mark up language (HTML) template files 104 with placeholders in markup text for reducing dynamic input data. An executable code module or data generation module 106 generates variable data, e.g., from system monitored parameters, which may be stored in input data store 108. Preferably, input data store 108 is non-volatile storage, although, any suitable volatile storage may be used as well. A second executable code module or page generation module 110 combines the variable data with appropriate HTML template files 104 to produce complete HTML documents 112, e.g., as described in U.S. Patent Application No. 09/852,959 entitled "System and Method for Improving The Performance of a Web Application by Building Only Requested Web Pages Only in A Requested Human Language" to Brewer et al., filed May 10, 2001, published November 14, 2002 as Published Application No. 2002/0169802, assigned to the assignee of the present invention and incorporated herein by reference The HTML documents 112 are transferred to web-based clients 114 using standard hypertext transport protocol (http) and displayed as web pages.

[0017] The design groups 102A, 102B, 102C independently and simultaneously design and/or modify of each of the HTML template files 104 and the data and page generation modules 106, 110, respectively. Thus, for this example, a development team may have responsibility split with groups 102B and 102C responsible for developing the two executable code modules 106, 110 and group 102A generating the HTML template files 104. As design group 102A designs new or updates old HTML templates, the HTML template files are passed to a pre-compiler 116. The pre-compiler 116 converts the HTML template files 104 to source and header files 117, preferably, in C language, which are passed to a compiler 118. Similarly, design teams 102B and 102C provide source code files 119 to the compiler 118 with the two executable code modules 106, 110, also, preferably, C language files. The compiler 118 combines the pre-compiler output files 117 with the source code files 117 to generate the data generation module 106 and the page generation module 110.

[0018] Thereafter during operation, the data generation module 106 receives and formats raw system data and stores it, locally in data store 108. Once a request is placed for the data, e.g., selecting the web address for the particular HTML document, the page generation module 110 selects the appropriate HTML template 104 and retrieves the corresponding data from the local data store 108. Then, the page generation module 110 combines the data in data store 108 with the HTML templates 104 to create HTML documents 112 for display 114.

[0019] The development groups 102A, 102B and 102C must necessarily maintain good communications. The data generation module 106 must provide whatever variable data is necessary and in the proper format. The page generation module 110 must accept the variable data and combine it with the HTML template files 104 in the correct placeholder locations to generate useful pages. The pre-compiler 116 facilitates this communication, operating on the HTML template files 104 to extract the placeholder names and produce a C language source code file and a header file that may then be used in compiling the

data and page generation modules 106, 110. Once design is complete, the compiler 118 and pre-compiler are no longer necessary and the final web-based management interface includes the data generation module 106 generating variable data and stored in input data store 108 and which the page generation module 110 combines appropriate HTML template files 104 to produce complete HTML documents 112 for display 114.

[0020] Figure 2 shows a prior data structure example 120 of how variable data was organized, e.g., which the executable modules combined with HTML files building system web pages for display such as described in Brewer et al. In this data structure example 120 variable data was organized within the executable code modules, e.g., 106, 110 in Figure 1, and stored as the dynamic input data for subsequent display as tabular data used in developing a typical state of the art web-based management interface. A page pointer table 122 in each executable code module 106, 110 includes links to repeated item structures 124, 126 in the pre-compiler output 117, e.g., in C language source and header files that the pre-compiler 116 may generate. The page pointer table 122 includes a page entry 128, 130, 132, 134 for each web page. Each page entry 128, 130, 132, 134 includes a pair of entries 136P, 136L, 138P, 138L, 140P, 140L, 142P, 142L, 144P, 144L for each of a number (N, N = 5 in this example) of defined tabular data lists, regardless of whether one, N or no lists are included in a particular web page. For each pair of list entries 136P, 136L, 138P, 138L, 140P, 140L, 142P, 142L, 144P, 144L, one entry (e.g., 140P) is a 32 bit pointer to a corresponding tabular data list 146, 148, 150, 152, 154 for the particular page. The second 16 bit entry (e.g., 140L) indicates the number of columns 156 in the particular tabular data list 146, 148, 150, 152, 154. Each page 128, 130, 132, 134 includes a list entry pair 136P, 136L, 138P, 138L, 140P, 140L, 142P, 142L, 144P, 144L, regardless of whether the particular web page includes a corresponding list or not, e.g., null/zero entry pair 144P, 144L.

[0021] So, in this example, each tabular data list 146, 148, 150, 152, 154 has similar information organized into number of columns determined by the corresponding second

entry 136L, 138L, 140L, 142L, 144L and a variable number of rows. The corresponding HTML template files 104 for this facility has placeholders embedded for each of the columns inside a pairing of start and end markers. For this simple example (i.e., N=5), the pre-allocated memory requirements places a significant constraint on the executable code. Each pair of entries 136P, 136L, 138P, 138L, 140P, 140L, 142P, 142L, 144P, 144L in this example requires six bytes in the code or 30 bytes for each page, even though many of the pages do not include tabular data lists. Since, typically, such an interface may exceed 200 pages, corresponding NULL/zero entry pair values are included, bloating data and page generation module 106, 110 by 6Kbytes. Further, expanding the number lists (e.g., N = 18) exacerbates this problem. For 213 pages, for example, increasing from 5 tabular data lists at 30 bytes per page to 18 at 108 further bloats the data and page generation module 106, 110 sizes from 6,390 bytes to 23,004 bytes.

[0022] Figure 3 shows a preferred data structure example 160 of variable data for tabular data lists 146, 148, 150, 152, 154, as organized within the executable code modules 106, 110 of Figure 1 and stored as the dynamic input data 108, in accordance with the present invention. By contrast, the page pointer table 162 has a single link 164, 166 to each repeated item list 124, 126 in dynamic input data store 104. Thus, each page entry 168, 170, 172, 174 has a single entry triplet 176P, 176M, 176N with the single page link 164, 166 indicated by entry triplet pointer 176P, again a 32 bit pointer. However, instead of individual links to each tabular data list 146, 148, 150, 152, 154, the page pointer table 162 includes a corresponding 32 bit pointer 176M to a page map 178, 180 with an 8 bit (for $N \leq 255$) number 176N indicating the number of page map entries. Each page map 178, 180, which are also in the pre-compiler output 117 with item structures 124, 126, includes an index 182, 184, 186, 188, 190 for each tabular data list 146, 148, 150, 152, 154. In this example, each map index entry is 16-bits, with twelve bits representing an offset into the data relative to entry triplet pointer 176P for the corresponding single page link 164, 166 and, the remaining four bits indicating the number of columns in the

respective tabular data list 146, 148, 150, 152, 154 for the individual repeated item. So, as can be seen from the example of Figure 3, the offset is zero for the first tabular data or repeated item list, 182, 190 on each page and the length is indicated by the non-zero digit. For the second and subsequent repeated item lists 184, 186, 188 on a page, the offsets are cumulative so that, the offset for the entry is added to the sum of the previous offsets and the number of columns for each.

[0023] Thus, instead of storing a pointer to the beginning of a tabular list for each repeated item structure and an integer to describe the structure's width, each page is described by an individual map 178, 180, which is a single data structure holding all of the information (offset pointers and column numbers) necessary to describe all of the repeated item structures on a given web page. Further, by storing the page maps 178, 180, e.g., with dynamic data in store 104, the page pointer table 162, which points to each map 178, 180, provides a much more efficient master data structure, requiring 36 bytes in the executable modules to describe the same structure described in Figure 2. Each additional page only increases executable module size by just nine (9) bytes to describe an essentially arbitrary number of repeated item lists on that additional page. Advantageously, for the above example of 213 pages (regardless of whether there are 5 or 18 possible tabular data lists) the code expands only by 1917 bytes instead of 6K to 23K. Further, for any page not including repeated item structures or lists, as is typical of most of the pages, there will only be nine wasted bytes for a null/null/zero triplet, instead of N*6, i.e., 30 for the example of Figure 2.

[0024] The pre-compiler conveniently and seamlessly adapts the same unmodified source code used for the executable modules from the data structure 120 of Figure 2 to a preferred structure as in the example 160 of Figure 3. The tabular data lists can be extracted and generated quickly and easily, shifting as indicated by the 4 bit length value and masking with the twelve offset bits to locate the page tables. Thus, advantageously, repeated item structures are stored in system memory and addressed in the executable

code for more efficient system resource use and for a more dynamic coding effort. Designers face fewer restrictions and so, have significantly more freedom in designing HTML template and in particular in incorporating tabular data into the web pages.

[0025] Advantageously, because variable system data is structured to minimize the impact of adding tables and pages with tables on code used in generating the pages, teams can now work together of a project much more easily than was heretofore possible. Source code developers, teams 102B and 102C, no longer need make difficult and risky changes to the operation of their code just support page layout decisions made by team 102A. Page layout team 102A no longer needs to avoid making page layout changes because such changes are no longer met with resistance to such changes from code developer teams 102B and 102C. As a result, the teams 102A, 102B, 102C can dedicate most of their time working freely on that aspect of the system in which each is a specialist, creating a final system that is of higher quality than would have otherwise been produced.

[0026] While the invention has been described in terms of preferred embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.